MULTIMEDIA UNIVERSITY

STUDENT ID NO

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2, 2015/2016

ECE4236 – PARALLEL PROCESSING & PROGRAMMING

(CE, EE, LE, ME & TE)

3 MARCH 2016
9:00 A.M – 11:00 A.M.
(2 Hours)

INSTRUCTIONS TO STUDENT

1. This question paper consists of 8 pages only (including this page).
2. **This is an open-book exam**, so the questions are more in-depth and there are fewer questions than normal exams.
3. There are **THREE (3) QUESTIONS** in this paper. **Answer ALL questions**. All questions carry equal marks (20 marks) and the distribution of the marks for each question is given.
4. All questions should be answered using the C programming language.
5. Write your answers in the Answer Booklet provided.
6. State all assumptions clearly.

Question 1

An entry wise exclusive OR (XOR) operation between two matrices of equal size is denoted as

$$\mathbf{A} \oplus \mathbf{B} = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0(n-1)} \\ a_{10} & a_{11} & \cdots & a_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(m-1)0} & a_{(m-1)1} & \cdots & a_{(m-1)(n-1)} \end{bmatrix} \oplus \begin{bmatrix} b_{00} & b_{01} & \cdots & b_{0(n-1)} \\ b_{10} & b_{11} & \cdots & b_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ b_{(m-1)0} & b_{(m-1)1} & \cdots & b_{(m-1)(n-1)} \end{bmatrix}$$

$$= \begin{bmatrix} a_{00} \oplus b_{00} & a_{01} \oplus b_{01} & \cdots & a_{0(n-1)} \oplus b_{0(n-1)} \\ a_{10} \oplus b_{10} & a_{11} \oplus b_{11} & \cdots & a_{1(n-1)} \oplus b_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(m-1)0} \oplus b_{(m-1)0} & a_{(m-1)1} \oplus b_{(m-1)1} & \cdots & a_{(m-1)(n-1)} \oplus b_{(m-1)(n-1)} \end{bmatrix} \quad (1.1)$$

Figure Q1-1 illustrates a partially completed code listing of the XOR operation between matrices **A** and **B**. Using the C programming language with a parallel programming implementation using **Windows (Win32) thread**

(a) Complete the `matrixXORFunc()` thread function in Figure Q1-2, which realizes a matrix based XOR operation as denoted in Eq. (1.1),with the following requirements:

i. The thread function computes the workload distribution based on a **tiled based (or *submatrix*)** approach. Here, the matrix, **C**, is segmented into a set of tiles, which is processed by threads. The maximum number of column divisors is two, regardless of the size of the matrix. However, the number of row divisors is calculated based on the number of rows in matrix, **C**, in line with the number of available threads. The thread function then executes the XOR operation based on the computed workload.

ii. Use the declared global variables in Figure Q1-2 to assist you in completing the definition of `matrixXORFunc()`.

iii. Note that eight (8) threads are created to parallelize the XOR operation. All of these threads execute the same thread callback function.

[12 marks]

(b) Complete the `main()` function Figure Q1-2 with the following requirements.

i. Create eight (8) threads, which execute the `matrixXORFunc()`. Initialize the necessary thread handle and thread identifier arrays, and then create the threads.

ii. The `main()` function waits for all threads to complete before exiting.

iii. You can assume that matrices **A**, **B** and **C** have been initialized. You can also assume that the content of matrices **A** and **B** have been populated with arbitrary data. There is no need to perform any code cleanup.

[4 marks]

Continued...

(c) Rewrite part (b) using OpenMP with dynamic scheduling. You should be able to identify and apply the correct type of OpenMP constructs into your solution. Specify parameters (or variables) that are to be shared or privatized.

Note: There is no need to re-write the dynamic memory creation and data reading from the text file in the `main()` *function. Focus on developing the* ***OpenMP*** *constructs for the iterative processes in the* `main()` *function. Include the appropriate header files in your program.*

[4 marks]

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <Windows.h>

#define NUM_THREADS 8

// Global variables
int g_RowC = 0, g_ColC = 0;
int g_CommonPoint = 0;
int **g_ppMatrixA = NULL;
int **g_ppMatrixB = NULL;
int **g_ppMatrixC = NULL;

// Part (a) - To be completed.
DWORD WINAPI matrixXORFunc(void *param);

// Main function
int main(){

      //Part (b) - To be completed.
      return 0;
}
```

Figure Q1-2: Partial program listing for XOR operation.

Continued...

Question 2

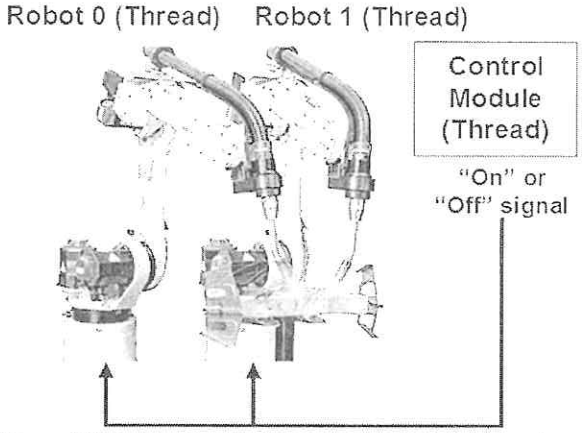

Figure Q2-1: Dual robotic arm management system.
Image source: http://www.lincolnelectric.com/en-us/

Figure Q2-1 illustrates an application simulator of a dual robotic arm management system. This system consists of a **control module** and **two industrial robotic arms**. Each robotic arm represents a thread, which receives a signal to either turn "on" or turn "off". When a turn "on" signal is received, the robotic arm prints out a turn "on" message on screen to confirm that it is turned on. When a turn "off" signal is received, the robotic arm prints out a turn "off" message on screen. Table Q2-1 describes the signal behaviour of the robotic arms.

Table Q2-1: Signal description for Figure Q2-1.

Signal (Decimal)	Robot 0	Robot 1
0	Off	Off
1	On	Off
2	Off	On
3	On	On

The sending of turn "on" and turn "off" signals to a robotic arm is controlled by the control module, which also represents a separate thread. In addition, Figure Q2-2 describes the partial code listing of this simulator. It includes the global variables and objects used by the control module and robot arm threads, the thread function prototypes (`ControlModuleFunc()` and `RobotFunc()`) and a partially completed `main()` function.

Using C programming language with a parallel programming implementation using **POSIX threads** and with reference to Figure Q2-2:

(a) Complete the thread callback function, `ControlModuleFunc()`, which simulates the behaviour of the control module.

i. This function continuously checks for changes to the `g_SignalVal` variable and triggers the corresponding "on" or "off" signal to the robot arm threads.

ii. This function also continuously checks if `g_SignalVal` is set to exit the

Continued...

thread function.

Note: *Since only one thread executes* `ControlModuleFunc()`*, you can ignore the function argument in* `ControlModuleFunc()`.

[6 marks]

(b) Complete the thread callback function, `RobotFunc()`, which simulates the behaviour of a robotic arm.

i. This function continuously checks for a turn "on" or "off" signal, in which this function prints out the result of the triggered signal on screen.
ii. This function also continuously checks for an exit signal triggered by the `ControlModuleFunc()` function, at which the thread function exits.

Note: *Use the function argument of* `RobotFunc()` *to determine the corresponding thread robotic arm identifier.*

[5 marks]

(c) Complete the `main()` function, which test the creation and functionality of the control module and robotic arm threads.

i. Write the code to initialize the mutexes and conditional variables used in the program.
ii. Write the code to create the control module and robotic arm threads.
iii. Complete `case -1` of the `switch(choice)` statement. Here, the control module and robotic arm threads are signalled to exit.
iv. Complete `case 0` to `case 3` of the `switch(choice)` statement. Here, `g_SignalVal` is updated based on the value of the `choice` variable.

[9 marks]

Note for parts (a), (b) and (c): *There is <u>no need</u> to re-write the complete program listing of Figure Q2-2 into your answer sheet. Focus on the specific sections of the code that is to be completed. You <u>need not</u> perform any code clean up.*

Continued...

```
#include "PThread\\pthread.h"

#define NUM_ROBOT_THREADS 2
pthread_mutex_t g_CRSignal;
int g_SignalVal = -1;

pthread_cond_t g_CMEvt;
pthread_mutex_t g_CMMutex;
int g_RobotOnOffArr[NUM_ROBOT_THREADS];

pthread_cond_t g_RobotEvtArr[NUM_ROBOT_THREADS];
pthread_mutex_t g_RobotMutexArr[NUM_ROBOT_THREADS];

// Part (a)(i) & (a)(ii) - To be completed.
void* ControlModuleFunc(void* pArg){}

// Part (b)(i) and (b)(ii) - To be completed.
void* RobotFunc(void* pArg){}

int main(){
      pthread_t hCMThread;
      pthread_t hRobotThreads[NUM_ROBOT_THREADS];
      int RobotThreadIDs[NUM_ROBOT_THREADS];

      // Part (c)(i) & (c)(ii) - To be completed.

      puts("\tMenu");
      puts("\t[0] Turn OFF both Robotic Arms");
      puts("\t[1] Turn ON Robotic Arm 0 only");
      puts("\t[2] Turn ON Robotic Arm 1 only");
      puts("\t[3] Turn ON both Robotic Arms");
      puts("\t[-1] Exit");
      int choice = 0;
      while(choice != -1){
            printf("Choice>> ");
            scanf("%d", &choice);
            switch(choice){
                  case -1:{
                        // Part (c)(iii) - To be completed.
                        break;
                  }
                  case 0:
                  case 1:
                  case 2:
                  case 3:{
                        // Part (c) (iv) - To be completed.
                        break;
                  }
                  default:{
                        break;
                  }
            }
      }
}
```

Figure Q2-2: Partial program listing for the robotic arm application simulator.

Continued...

Question 3

The complex conjugate of a complex number is the number with equal real part and imaginary part equal in magnitude but opposite in sign. Mathematically, the complex conjugate of $z = a + bi$ is defined as $z^* = a - bi$, where a and b are the real and imaginary values, respectively.

The multiplication of complex number with its conjugate will yield a real number,

$$\mathrm{R} = z \times z^* = (a + bi)(a - bi) = a^2 - b^2 i^2 \qquad (3.1)$$

Figure Q3-1 illustrates the content of a text file, **complex.txt**, which contains a list of complex numbers and corresponding complex conjugates. The first row in **complex.txt** represents the number of complex numbers in this text file. The second row represents the legend text and the third row onwards lists the complex numbers (z) and corresponding complex conjugates (z^*). In detail, from the third row onwards, the first two columns represent the real and imaginary values of z and the last two columns represent the real and imaginary values of z^*.

Obviously, not all of the complex numbers in this text file have a correct conjugate assignment. For instance, the conjugate of the second complex number in this text file (i.e., $z = 3.2 + 7.2i$) is $z^* = -3.2 - 7.2i$, which is incorrect. Instead, the correct conjugate for this complex number should be $z^* = 3.2 - 7.2i$. A similar error is also observed for the fourth complex number in this text file. Other equivalent errors are also possible in this text file, though not illustrated in Figure Q3-1.

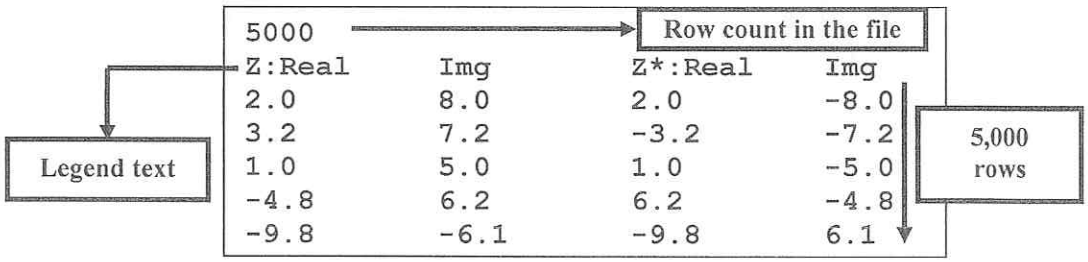

Figure Q3-1: Content of complex.txt

Using the C programming language with a parallel programming implementation using the **Message Passing Interface (MPI),** write a program to:

1) Identify and extract the correct conjugate assignment of the complex numbers in **complex.txt** into a separate text file.
2) Implement $z \times z^*$ for the complex numbers with the correct conjugates.

with the following requirements:

(a) Only the **root rank** (or **first node**) is permitted to access the "**complex.txt**" file.

i. The root rank reads z and z^* of each row from "**complex.txt**" file and determines if z^* is in fact an actual conjugate of z.

ii. The root transmit the results of determination process and the coefficients of z to the subsequent node.

[9 marks]

Continued...

(b) Continuing from part (a), the **second node** receives results of determination process and the coefficients of z (per row) from the first node. Then, this node computes $z \times z^*$ if z^* is in fact an actual conjugate of z.

This node transmit the results of determination process, the coefficients of z and the outcome of $z \times z^*$ to the third node.

[6 marks]

(c) Continuing from part (b), the **third node** receives the results of determination process, the coefficients of z and the outcome of $z \times z^*$ (per row) from the second node.

This node creates a new text file, "**complex_correct.txt**". This node writes the correct z, z^* and $z \times z^*$ into this text file, based on a sample illustration as seen in Figure Q3-2.

[5 marks]

Note: Write parts (a), (b) and (c) as a single parallel program. Use a **parallel pipeline structure.** The total number of nodes in this pipeline structure should be three as described in parts (a), (b) and (c). Include the necessary header files to support the implementation of this program.

```
5000
Z:Real      Img         Z*:Real     Img
2.0         8.0         2.0         -8.0
1.0         5.0         1.0         -5.0
-9.8        -6.1        -9.8        6.1
```

Figure Q3-2: Content of complex_correct.txt

End of Paper